Transparent Data Encryption (TDE)

WHY NEED Transparent Data Encryption (TDE)

Oracle Database uses authentication, authorization, and auditing mechanisms to secure data in the database, but not in the operating system data files where data is stored.

To protect these data files, Oracle Database provides Transparent Data Encryption (TDE).

Transparent Data Encryption (TDE) enables to encrypt sensitive data, such as Personally Identifiable Information (PII), Credit card number store in tables and tablespaces

After the data is encrypted, this data is transparently decrypted for authorized users or applications when they access this data.

Even if the encrypted data is retrieved, it cannot be understood until authorized decryption occurs, which is automatic for users authorized to access the table.

Both TDE column encryption and TDE tablespace encryption use a two-tiered key-based architecture.

Unauthorized users, such as intruders who are attempting security attacks, cannot read the data from storage and back up media unless they have the TDE master encryption key to decrypt it.

Encryption Components

Two-tier key architecture

Both **column and table space** encryption keys are stored in the database but are encrypted with another key called the master key

The master key is stored outside the database in a special container called an External security module

Master key is stored in **Oracle wallet**

Unless the right password is supplied, the wallet can't be opened and the encrypted data can't be retrieved. The wallet is automatically closed when the database instance is shut down and must be reopened by a security officer when the instance starts

Thieves might be able to restore a database from tapes, without the wallet and the password, they will not be able to view the encrypted data.

Algorithm

DES (Triple Data Encryption Standard)

AES (Advanced Encryption Standard)

AES128 is default for tablespace

AES192 is default for column encryption

Encryption key

Master encryption key – The encryption key used to encrypt secondary data encryption keys used for column encryption and table space encryption. Master encryption keys are part of the Oracle Advanced Security two-tier key architecture

Stores

Wallet – A PKCS#12 archiving file format is used to stores TDE master key. Or
PKCS #5 is the Password-Based Cryptography Specification.

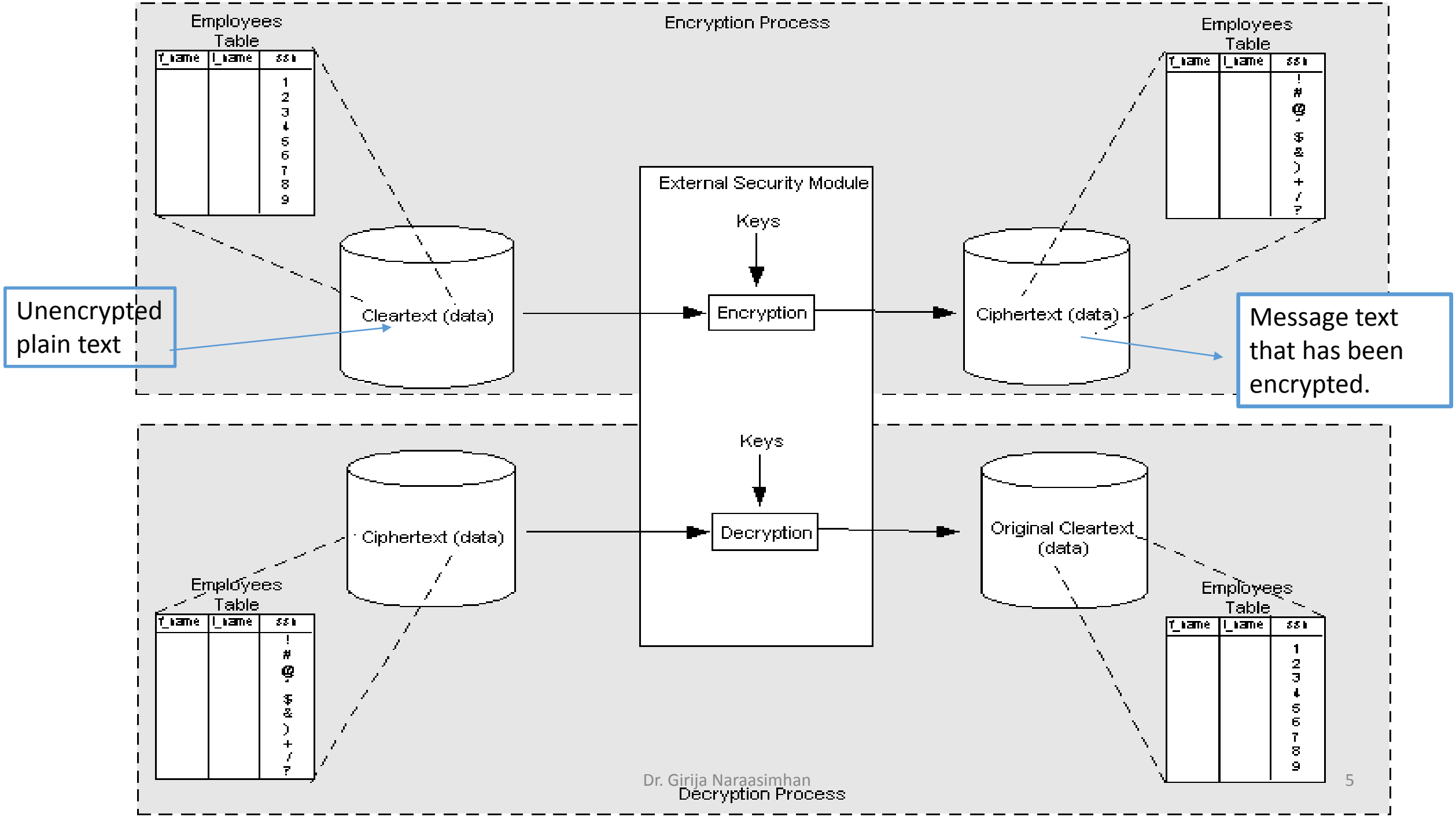
Column encryption

Table key – it is also referred as **column key**, this key is used to encrypt one or more specific columns in a given table.

These keys are stored in the **Oracle data dictionary**, encrypted with the master encryption key.

Table space encryption

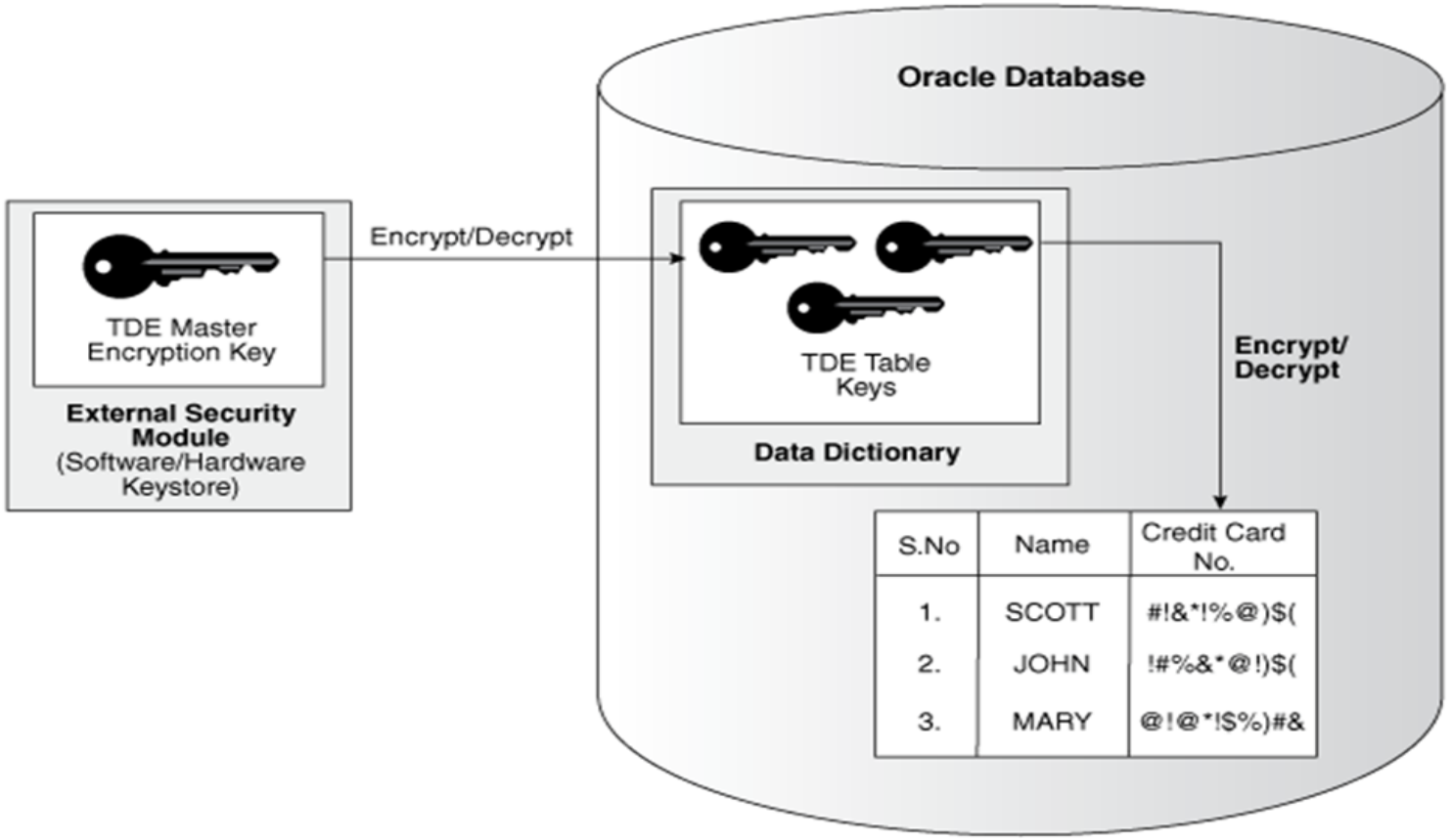
The key used to encrypt a tablespace. These keys are encrypted using the **master key** and are stored in the **tablespace header of the encrypted tablespace**, as well as in the **header of each operating system** - file that belongs to the encrypted tablespace.


Employees Table
f_name
l_name
ssn
1
2
3
4
5
6
7
8
9
Encryption Process
Employees Table
f_name
l_name
ssn
! # @ . $ &) + / ?
External Security Module
Keys
Encryption
Cleartext (data)
Ciphertext (data)
Unencrypted plain text
Message text that has been encrypted.
Keys
Decryption
Ciphertext (data)
Original Cleartext (data)
Employees Table
f_name
l_name
ssn
! # @ . $ &) + / ?
Employees Table
f_name
l_name
ssn
1
2
3
4
5
6
7
8
9
Decryption Process

the master key of the server is stored in an external security module that is outside the database and accessible only to the security administrator.

For this external security module, Oracle uses an Oracle wallet Storing the master key in this way prevents its unauthorized use.

In addition to storing the master key, the Oracle wallet is also used to generate encryption keys and perform encryption and decryption.



When a table contains encrypted columns, a single key is used **regardless of the number of encrypted** columns.

The keys for all tables containing encrypted columns are encrypted with the database server master key and stored in a dictionary table in the database.

Specifying an Additional Wallet Location in SQLNET.ORA

By default, the external security module stores encryption keys in the Oracle wallet specified in the sqlnet.ora configuration file.

If no wallet location is specified in the sqlnet.ora file, then the default database wallet is used.

If you wish to use a wallet specifically for transparent data encryption, then you must specify a second wallet location in sqlnet.ora by using the ENCRYPTION_WALLET_LOCATION parameter.

```
ALTER SYSTEM SET ENCRYPTION KEY certificate_ID IDENTIFIED BY password
```

certificate_ID is an optional string containing the unique identifier of a certificate stored in the security module. Use this parameter if you intend to use your PKI private key as your master key. This parameter has no default setting.

You can search for a certificate_ID by querying the V$WALLET fixed view when the wallet is open. Only certificates that can be used as master keys by transparent data encryption are shown.

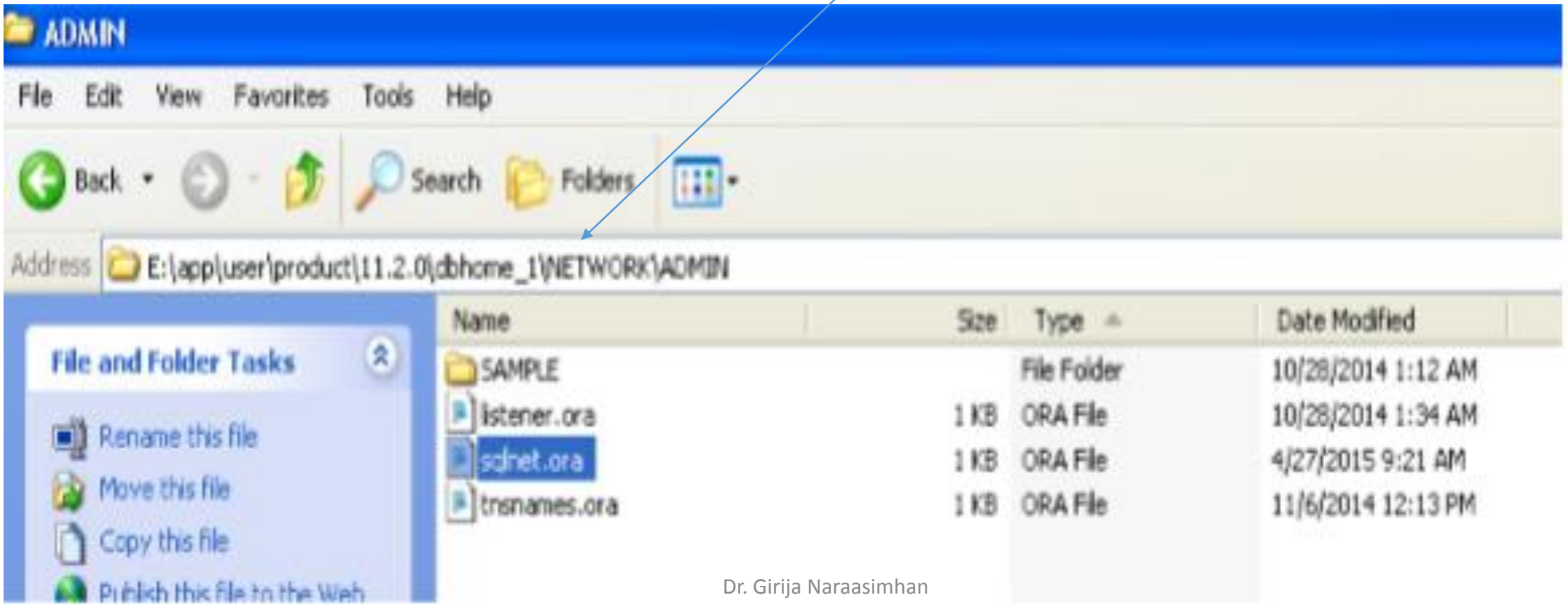
password is the mandatory wallet password for the security module, with no default setting. It is case sensitive; enclose it in double-quotation marks.

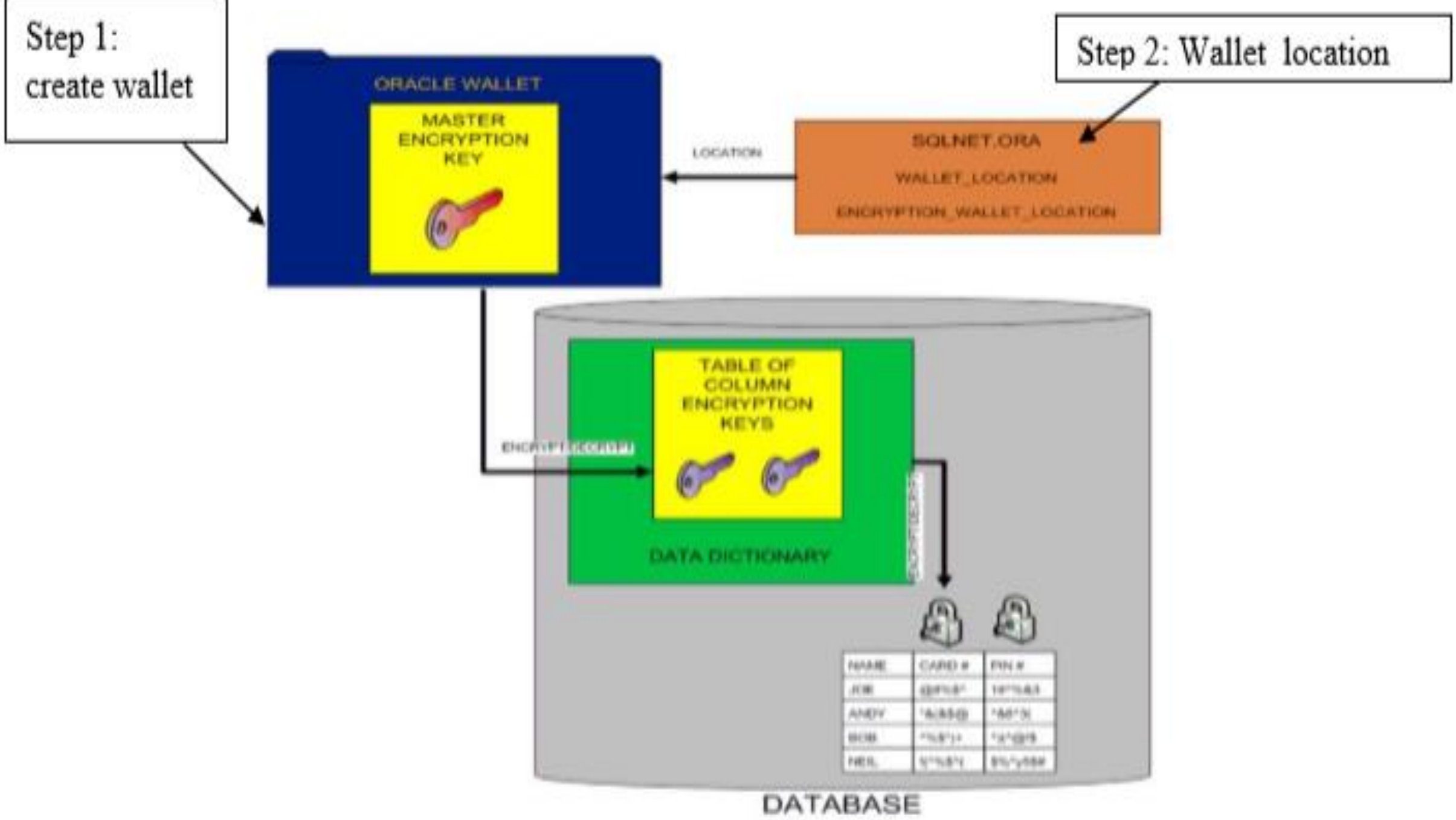
Step 1: Create Wallet E:\app\user\admin\orcl\wallet – in the orcl directory create “Wallet” directory suppose it is not available.

Step 2: insert wallet location in “sqlnet.ora”, as given below

```
ENCRYPTION_WALLET_LOCATION=
(SOURCE=(METHOD=FILE)(METHOD_DATA=
  (DIRECTORY=E:\app\user\admin\orcl\wallet)))
```

In the directory give correct path where you create the wallet directory.




Step 1:
create wallet
Step 2: Wallet location
ORACLE WALLET
MASTER
ENCRYPTION
KEY
LOCATION
SQLNET.ORA
WALLET_LOCATION
ENCRYPTION_WALLET_LOCATION
TABLE OF
COLUMN
ENCRYPTION
KEYS
DATA DICTIONARY
NAME
CARD #
PIN #
ANDY
DATABASE

Create New Master key

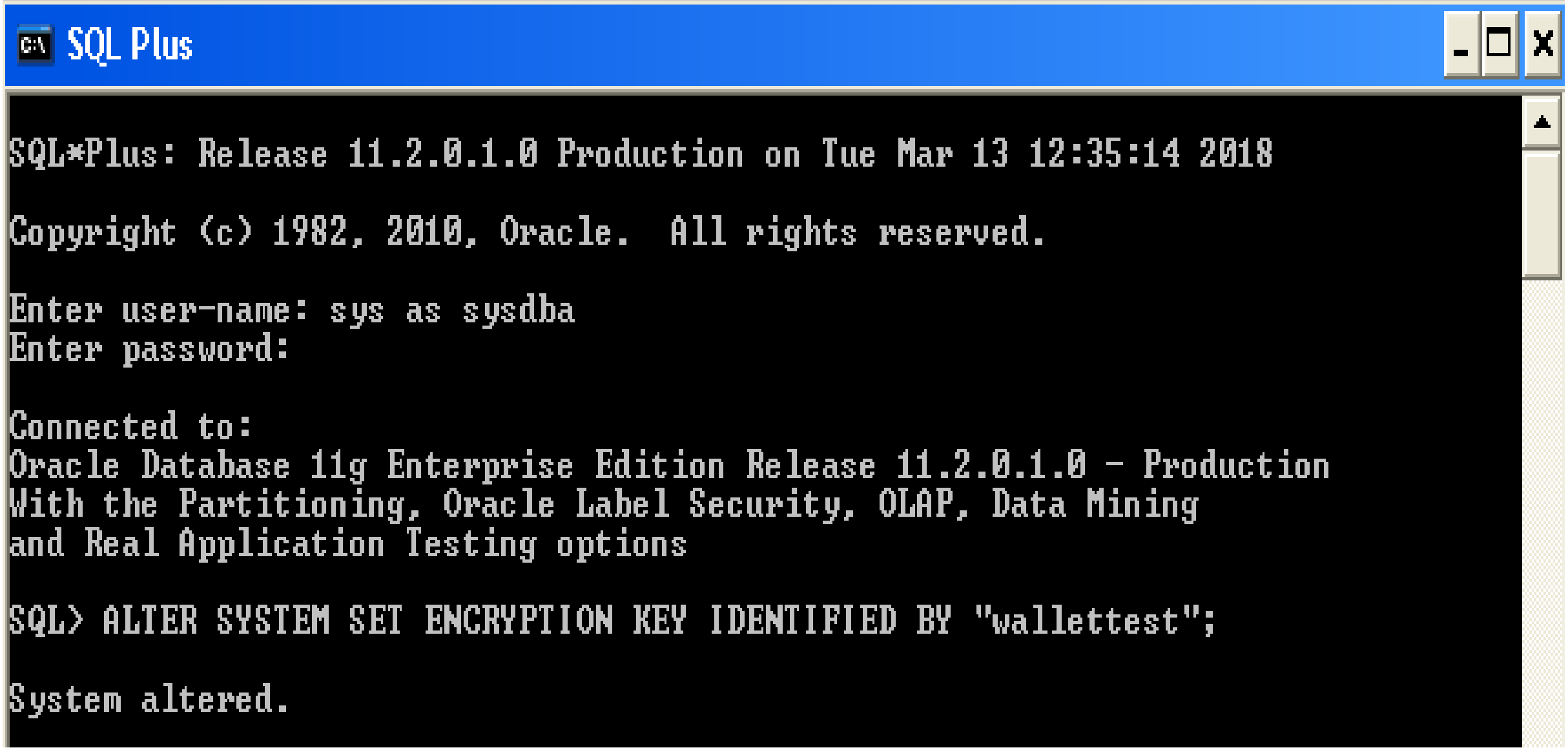
To use transparent data encryption, it is needed ALTER SYSTEM privilege and a valid password to the Oracle wallet.

If an Oracle wallet does not exist, then a new one is created using the password specified in the SQL command.

To create a new master key and begin using transparent data encryption

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "password";
```

ALTER SYSTEM SET ENCRYPTION KEY command is a DDL command requiring the ALTER SYSTEM privilege, and it automatically commits any pending transactions.


SQL Plus
SQL*Plus: Release 11.2.0.1.0 Production on Tue Mar 13 12:35:14 2018
Copyright (c) 1982, 2010, Oracle. All rights reserved.
Enter user-name: sys as sysdba
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, Oracle Label Security, OLAP, Data Mining
and Real Application Testing options
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "wallettest";
System altered.

Open the Wallet

```
Alter system set encryption wallet open identified by "wallettest";
```

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "wallettest";

System altered.
```

Once the wallet has been opened, it remains open until you shut down the database instance

or

close it explicitly by issuing an ALTER SYSTEM SET ENCRYPTION WALLET CLOSE command.

When you restart the instance, you must issue the ALTER SYSTEM SET ENCRYPTION WALLET OPEN command again.

Status of the Wallet

```
select wrl_type wallet,status,wrl_parameter wallet_location from v$encryption_wallet;
```

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "wallettest";

System altered.

SQL> select wrl_type wallet,status,wrl_parameter wallet_location from v$encrypti
on_wallet;

WALLET               STATUS
-------------------- ------------------
WALLET_LOCATION
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------
file                 OPEN
E:\app\user\admin\orcl\wallet
```

If the wallet is already open, the command returns an error and takes no action

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "wallettest";
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "wallettest"
*
ERROR at line 1:
ORA-28354: wallet already open
```

Close the Wallet

Close the wallet using the below given command

```
Alter system set encryption wallet close identified by "wallettest";
```

```
SQL> alter system set wallet close identified by "wallettest";

System altered.
```

```
SQL> alter system set wallet close identified by "wallettest";

System altered.

SQL> select wrl_type wallet,status,wrl_parameter wallet_location from v$encrypti
on_wallet;

WALLET               STATUS
-------------------- ------------------
WALLET_LOCATION
--------------------------------------------------------------------------------

file                 CLOSED
E:\app\user\admin\orcl\wallet
```

Incorrect password

If the schema does not have the ALTER SYSTEM privilege, or the wallet is unavailable, or an incorrect password is given, then the command returns an error and exits.

Instead of “wallettest” try with incorrect password “wtest”

```
SQL> alter system set encryption wallet open identified by "wtest";
alter system set encryption wallet open identified by "wtest"
*
ERROR at line 1:
ORA-28353: failed to open wallet


SQL> alter system set encryption wallet open identified by "wallettest";

System altered.
```

Using Wallets with Automatic Login Enabled for Transparent Data Encryption

The external security module can use wallets with the automatic login feature enabled.

These wallets remain open all the time.

The security administrator does not have to reopen the wallet after a database instance has been restarted.

If your environment does not require the extra security provided by a wallet that must be explicitly opened for use, then you may use an auto login wallet.

Creating a Table with an Encrypted Column

To create a new table with encrypted columns, use the CREATE TABLE command in the following form:

CREATE TABLE *table_name* (*column_name column_type* ENCRYPT,....);

The ENCRYPT keyword against a column specifies that the column should be encrypted.

By default, transparent data encryption uses AES with a 192-bit length key (AES192).

```
SQL> CREATE TABLE employee(first_name VARCHAR2(128), last_name VARCHAR2(128), empID NUMBER, salary N
UMBER(6) ENCRYPT);
CREATE TABLE employee(first_name VARCHAR2(128), last_name VARCHAR2(128), empID NUMBER, salary NUMBER
(6) ENCRYPT)
                                                                *

ERROR at line 1:
ORA-28336: cannot encrypt SYS owned objects


SQL> conn hr/hr
Connected.
SQL> CREATE TABLE employee(first_name VARCHAR2(128), last_name VARCHAR2(128), empID NUMBER, salary N
UMBER(6) ENCRYPT);

Table created.
```

```
CREATE TABLE employee
(first_name VARCHAR2(128),
last_name VARCHAR2(128),
empID NUMBER,
salary NUMBER(6) ENCRYPT );
```

Sys objects are not allow to encrypted.

```
SQL> conn hr/hr
Connected.
SQL> CREATE TABLE employee(first_name VARCHAR2(128), last_name VARCHAR2(128), empID NUMBER, salary N
UMBER(6) ENCRYPT);

Table created.

SQL>
SQL> insert into employee values('INDUJA','VANAJA',1234,500);

1 row created.

SQL> select * from employee;

FIRST_NAME
                         LAST_NAME
                                                             EMPID     SALARY
_____________________________________________________________________________
_____________________________ ___________________________________________________
_____________________________________________ __________ __________
INDUJA
                         VANAJA
                                                              1234        500

SQL> select salary from employee;

    SALARY
__________
       500

SQL> commit;

Commit complete.

SQL> select salary from employee;

    SALARY
__________
       500
```

```
SQL> conn sys as sysdba
Enter password:
Connected.
SQL> Alter system set encryption wallet close identified by "wallettest";

System altered.

SQL> select salary from hr.employee;
select salary from hr.employee
                     *
ERROR at line 1:
ORA-28365: wallet is not open


SQL> select first_name from hr.employee;

FIRST_NAME
--------------------------------------------------------------------------------
--------------------------------
INDUJA

SQL>
SQL> alter system set encryption wallet open identified by "wallettest";

System altered.

SQL> select salary from hr.employee;

    SALARY
----------
       500
```

If wallet is open, then encrypted column will display.

If wallet is closed and select the encrypted column salary, it will not display the column , but other column (clear text) column same tables are displaying.

Salt

Salt is a way to strengthen the security of encrypted data.

It is a random string added to the data before it is encrypted, causing repetition of text in the clear to appear different when encrypted.

Salt thus removes one method attackers use to steal data, namely, matching patterns of encrypted text.

By default, transparent data encryption adds salt to clear text before encrypting it.

This makes it harder for attackers to steal the data through a brute force attack.

Adding Salt to an Encrypted Column

```
ALTER TABLE employee MODIFY (first_name ENCRYPT SALT);
```

```
SQL> conn hr/hr
Connected.
SQL> ALTER TABLE employee MODIFY (first_name ENCRYPT SALT);

Table altered.

SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       FIRST_NAME                     YES
EMPLOYEE                       SALARY                         YES
```

Display the salt and encrypted column details

```
select table_name,column_name,salt from user_encrypted_columns;
```

Removing Salt from an Encrypted Column

```
ALTER TABLE employee MODIFY (first_name ENCRYPT NO SALT);
```

```
SQL> ALTER TABLE employee MODIFY (first_name ENCRYPT NO SALT);

Table altered.

SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       FIRST_NAME                     NO
EMPLOYEE                       SALARY                         YES
```

Encrypted Column Using No salt for Indexing

Index the encrypted column, must specify the NO SALT parameter with the SQL ENCRYPT clause

```
SQL> ALTER TABLE employee MODIFY (first_name ENCRYPT NO SALT);

Table altered.

SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       FIRST_NAME                     NO
EMPLOYEE                       SALARY                         YES

SQL> create index firstind on employee(first_name);

Index created.

SQL> create index salind on employee(salary);
create index salind on employee(salary)
                                *
ERROR at line 1:
ORA-28338: Column(s) cannot be both indexed and encrypted with salt
```

To remove salt from an encrypted column before indexing it,

Encryption Algorithms for TDE

Table 2-1 Supported Encryption Algorithms for Transparent Data Encryption

Algorithm	Key Size	Parameter Name
Triple Encryption Standard (DES)	168 bits	`3DES168`
Advanced Encryption Standard (AES)	128 bits	`AES128`
AES	• Default for column level encryption is 192 bits • Default for tablespace encryption is 128 bits	• `AES192` for column level encryption • `AES128` for tablespace encryption
AES	256 bits	`AES256`

For integrity protection of TDE column encryption, the `SHA-1` hashing algorithm is used. If you have storage restrictions, then use the `NOMAC` option.

By default, transparent data encryption uses AES with a 192-bit length key (AES192). AES128 is default for tablespace.

Change the different algorithm already encrypted column

```
SQL> create table employee(empid number(5) encrypt using '3DES168', fname varchar2(10));

Table created.

SQL> alter table employee modify(fname varchar2(10) encrypt using '3DES168');

Table altered.
```

Include other column also same algorithm then no problem, for example here fname also has same algorithm '3DES168'. if the column is already encrypted it is not possible to change the different algorithm

```
SQL> alter table employee modify(fname varchar2(10) encrypt using 'AES192');
alter table employee modify(fname varchar2(10) encrypt using 'AES192')
                            *
ERROR at line 1:
ORA-28334: column is already encrypted
```

All encrypted column should be same Algorithm

In the table, all the column should be in the same algorithm

```
SQL> alter table employee add(lname varchar2(10));

Table altered.

SQL> alter table employee modify(lname varchar2(10) encrypt using 'AES192');
alter table employee modify(lname varchar2(10) encrypt using 'AES192')
                            *
ERROR at line 1:
ORA-28340: a different encryption algorithm has been chosen for the table
```

Table with an Encrypted Column Using 3DES168

create table employee(empid number(5) encrypt using '3DES168', fname varchar2(10));

It also possible to assign the syntax for specifying a different encryption algorithm.

The string which specifies the algorithm must be enclosed in single quotation marks.

```
SQL> drop table employee purge;

Table dropped.

SQL>
SQL> create table employee(empid number(5) encrypt using '3DES168', fname varchar2(10));

Table created.
```

Display encrypted algorithm in the table

```
SQL> select table_name,column_name,encryption_alg from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    ENCRYPTION_ALG
------------------------------ ------------------------------ ------------------------------
EMPLOYEE                       EMPID                          3 Key Triple DES 168 bits key
EMPLOYEE                       FNAME                          3 Key Triple DES 168 bits key
EMPLOYEE                       SALARY                         3 Key Triple DES 168 bits key
```

Disabling Encryption on a Column

It may be necessary to disable encryption for reasons of compatibility or performance.

To disable column encryption, use the ALTER TABLE MODIFY command with the DECRYPT clause

```
SQL> ALTER TABLE employee MODIFY (fname DECRYPT);

Table altered.

SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       EMPID                          YES
```

Adding Encrypted Columns to Existing Tables

```
SQL> ALTER TABLE employee ADD (salary number(10) ENCRYPT);

Table altered.

SQL>
SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       EMPID                          YES
EMPLOYEE                       SALARY                         YES
```

Encrypting Unencrypted Columns

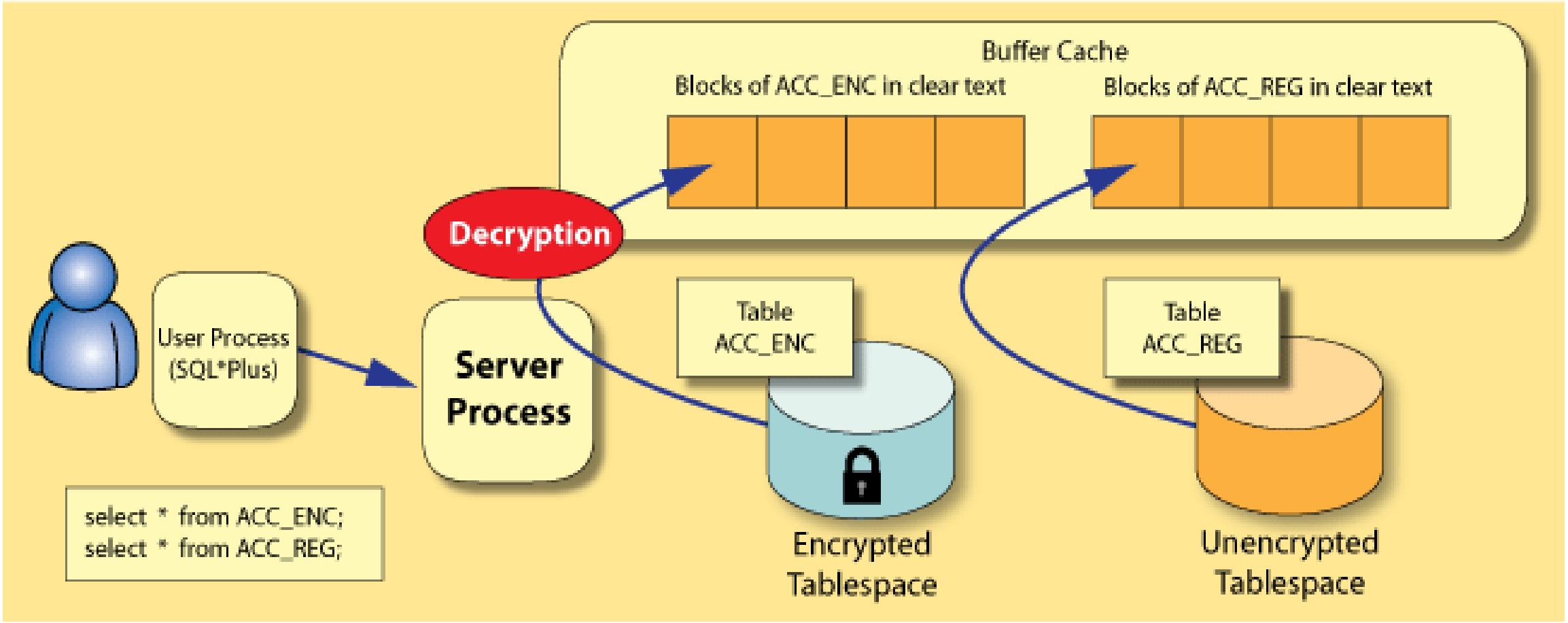
To encrypt unencrypted columns, use the ALTER TABLE MODIFY command, specifying the unencrypted column with the ENCRYPT clause

```
SQL> ALTER TABLE employee modify(fname ENCRYPT);

Table altered.

SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       EMPID                          YES
EMPLOYEE                       FNAME                          YES
EMPLOYEE                       SALARY                         YES
```



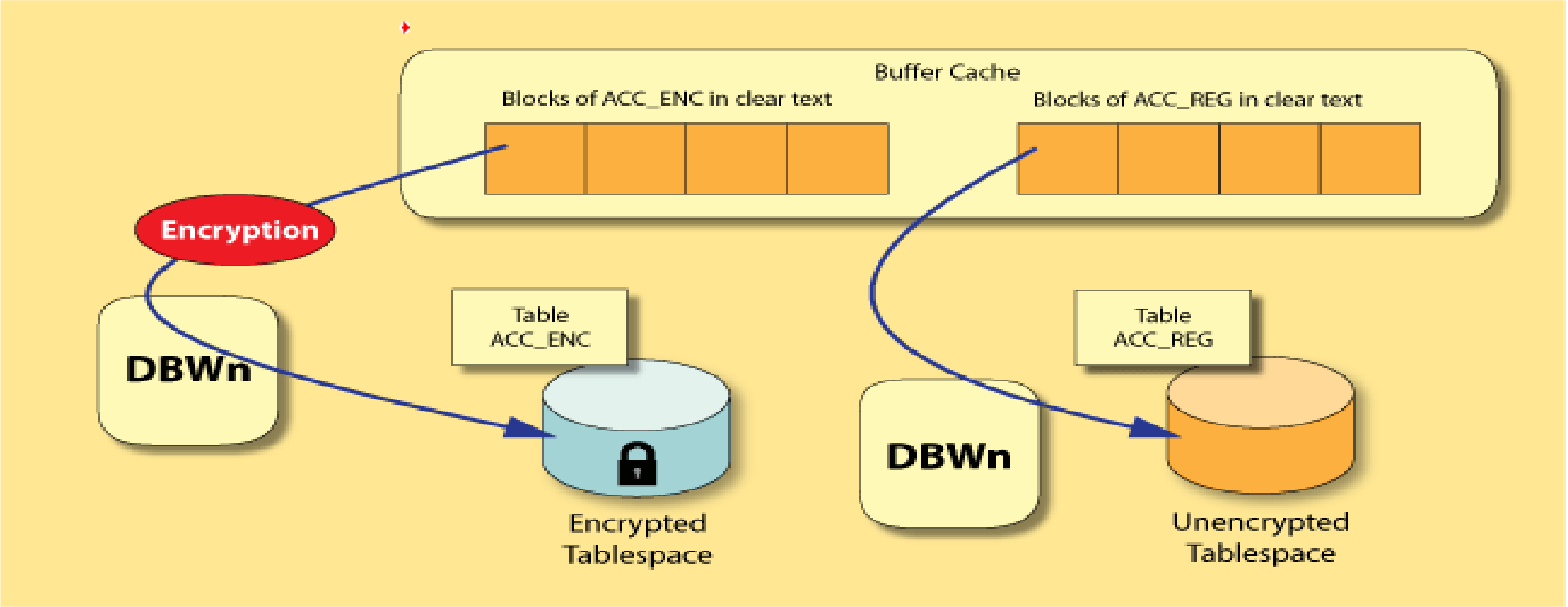
Loading of data buffers

With tablespace encryption, before data buffers are written back to disk (as a result of the checkpoint process), they are encrypted by DB Writer processes (DBWn).

Operations, such as direct path inserts and reads that manipulate the data directly in the database, perform encryption inline.

When the log buffers are written to the redo logs by the log writer process, they are encrypted as well.

so the initial and subsequently archived redo logs contain only encrypted data.



Flushing of buffers from cache to disk

Create an Encrypted Tablespace

TDE tablespace encryption enables you to encrypt an entire tablespace.

All data stored in the tablespace is encrypted by default

```
SQL> CONN SYS AS SYSDBA
Enter password:
Connected.
SQL> CREATE TABLESPACE encryptts
  2       DATAFILE 'c:\temp\encrts.dbf'
  3       SIZE 150M
  4       ENCRYPTION
  5       DEFAULT STORAGE(ENCRYPT);

Tablespace created.
```

Create a Table in an Encrypted Tablespace

If we create a table in an encrypted tablespace, then all data in the table is stored in encrypted form on the disk

```
SQL> CONN HR/HR
Connected.
```

```
SQL> create table employee123(empid number(10), fname varchar2(15)) tablespace encryptts;

Table created.

SQL> select table_name,column_name,salt from user_encrypted_columns;

TABLE_NAME                     COLUMN_NAME                    SAL
------------------------------ ------------------------------ ---
EMPLOYEE                       EMPID                          YES
EMPLOYEE                       FNAME                          YES
EMPLOYEE                       SALARY                         YES
```

```
SQL> DESC USER_ENCRYPTED_COLUMNS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TABLE_NAME                                NOT NULL VARCHAR2(30)
 COLUMN_NAME                               NOT NULL VARCHAR2(30)
 ENCRYPTION_ALG                                     VARCHAR2(29)
 SALT                                               VARCHAR2(3)
 INTEGRITY_ALG                                      VARCHAR2(12)
```

References

https://docs.oracle.com/cd/B19306_01/network.102/b14268/asotrans.htm#BABJJAIG

https://docs.oracle.com/cloud/latest/db121/ASOAG/asotrans.htm#ASOAG10136

http://www.oracle.com/technetwork/testcontent/o19tte-086996.html